In the Claims:

Please amend claims 2-8, 10-13, and 15-17. The claims are as follows:

1. (Previously presented) A method of determining, in a computer environment, the equivalence, if any, of two algebraic expressions for use in compiler optimisation of source code and like computing tasks, said method comprising the steps of:

(a) recasting said expressions into a form of one or more token pairs arranged sequentially in a string, each said token pair comprising an operator followed by an operand;

(b) reducing said strings in accordance with a set of predetermined simplifying rules;

(c) comparing the reduced strings by matching, to detect equivalence of the two algebraic expressions; and

(c1) compiling said source code into object code, wherein said source code comprises said two algebraic expressions, and wherein said compiling comprises said recasting, said reducing, and said comparing.

2. (Currently amended) The method of claim 1, ~~whereby~~ wherein the recasting step (a) is preceded by a preconditioning step comprising, in relation to said algebraic expressions, the following sub-steps according to whether a sub step applies:

(da) deleting a space in the expression;

(db) removing a bracket in the expression by expanding a bracketed sub-expression;

(dc) inserting a unitary operator at the start of the expression;

(dd) recasting a power factor, being a variable being raised to a power in the

expression, in an alternate form as one of:

(dda) the power factor being expressed as the variable multiplied by itself as many times as the power, if the power is a positive integer;

(ddb) the power factor being expressed as a reciprocal of the variable multiplied by itself as many times as an absolute value of the power, if the power is a negative integer;

(ddc) the power factor being replaced by an appropriate function which can compute the power factor, if the power is not an integer;

(de) recasting a constant in the expression in exponential format;

(df) substituting a "+" operator in the expression by "+1*", a "1" being in exponential format;

(dg) substituting a "-" operator in the expression by "-1*", a "1" being in exponential format; and

(dh) recasting a "division by a constant" in the expression as multiplication by a reciprocal of the constant, wherein said compiling comprises said preconditioning step.

3. (Currently amended) The method of claim 1, ~~whereby~~ wherein the simplifying rules in step (b) comprise:

(ba) arranging token pairs into subgroups;

(bb) arranging operand tokens in an arranged subgroup in order;

(bc) reducing the ordered operands by consolidating one or more constants and eliminating variables of opposite effect to form reduced subgroups; and

(bd) consolidating one or more multiple instances of similar subgroups, to produce a

reduced string.

4. (Currently amended) The method of claim 1, ~~whereby~~ wherein an algebraic expression whose equivalence is to be determined contains an aliased variable, said method further comprising the steps of:

arranging an ordered list of aliases of the variable, and substituting a first alias in the ordered list for all instances of the aliased variable in the expression, wherein said compiling comprises said arranging an ordered list of aliases of the variable and said substituting a first alias in the ordered list.

5. (Currently amended) The method of claim 1, ~~whereby~~ wherein an algebraic expression whose equivalence is to be determined contains a function, said method further comprising the steps of:

reducing function arguments using the set of predetermined simplifying rules; and

replacing the function by a tagged string, said string designating a function name, parameter types, and arguments, ~~whereby~~ wherein the tag distinguishes the function name from a variable, wherein said compiling comprises said reducing function arguments and said replacing the function by a tagged string.

6. (Currently amended) An apparatus adapted to determine, in a computer environment, the equivalence, if any, of two algebraic expressions for use in compiler optimisation of source code and like computing tasks, said apparatus comprising:

(a) recasting means for recasting said expressions into a form of one or more token pairs

arranged sequentially in a string, each said token pair comprising an operator followed by an operand;

(b) reduction means for reducing said strings in accordance with a set of predetermined simplifying rules;

(c) comparison means for comparing the reduced strings by matching, to detect equivalence of the two algebraic expressions; and

(c1) means for compiling said source code into object code, wherein said source code comprises said two algebraic expressions, and wherein said means for compiling comprises said recasting means for recasting, said reduction means for reducing, and said comparison means for comparing.

7. (Currently amended) A computer program product including a computer readable medium having recorded thereon a computer program for determining, in a computer environment, the equivalence, if any, of two algebraic expressions for use in compiler optimisation of source code and like computing tasks, said computer program comprising:

(a) recasting process steps for recasting said expressions into a form of one or more token pairs arranged sequentially in a string, each said token pair comprising an operator followed by an operand;

(b) reduction process steps for reducing said strings in accordance with a set of predetermined simplifying rules;

(c) comparison process steps for comparing the reduced strings by matching, to detect equivalence of the two algebraic expressions; and

(c1) compiling process steps for compiling said source code into object code, wherein said source code comprises said two algebraic expressions, and wherein said compiling process steps for compiling comprise[[s]] said recasting process steps for recasting, said ~~reducing~~ reduction process steps for reducing, and said ~~comparing~~ comparison process step for comparing.

8. (Currently amended) The method of claim [[1]] 7, wherein said reduction process steps comprise:

processing token pairs into an ordered arrangement;

determining ~~whether~~ that a redundant equivalent subexpression exists within said ordered arrangement; and

[[if]] responsive to said determining ~~determines~~ that said redundant equivalent subexpression exists within said ordered arrangement, ~~then~~ eliminating said redundant equivalent subexpression from said ordered arrangement.

9. (Canceled)

10. (Currently amended) The apparatus of claim 6, ~~whereby~~ wherein the simplifying rules in step (b) comprise:

(ba) arranging token pairs into subgroups;

(bb) arranging operand tokens in an arranged subgroup in order;

(bc) reducing the ordered operands by consolidating one or more constants and eliminating variables of opposite effect to form reduced subgroups; and

(bd) consolidating one or more multiple instances of similar subgroups, to produce a reduced string.

11. (Currently amended) The apparatus of claim 6, ~~whereby~~ wherein an algebraic expression whose equivalence is to be determined contains an aliased variable, said apparatus further comprising:

means for arranging an ordered list of aliases of the variable[[,]]; and

means for substituting a first alias in the ordered list for all instances of the aliased variable in the expression, wherein said means for compiling comprises said means for arranging an ordered list of aliases and said means for substituting a first alias in the ordered list.

12. (Currently amended) The apparatus of claim 6, ~~whereby~~ wherein an algebraic expression whose equivalence is to be determined contains a function, said apparatus further comprising:

means for reducing function arguments using the set of predetermined simplifying rules; and

means for replacing the function by a tagged string, said string designating a function name, parameter types, and arguments, ~~whereby~~ wherein the tag distinguishes the function name from a variable,

wherein said means for compiling comprises said means for reducing function arguments and said means for replacing the function.

13. (Currently amended) The apparatus of claim 6, wherein said reduction means for reducing

said strings comprises:

means for processing token pairs into an ordered arrangement;

means for determining whether a redundant equivalent subexpression exists within said ordered arrangement; and

means for eliminating said redundant equivalent subexpression from said ordered arrangement, wherein said means for compiling comprises said means for processing token pairs, said means for determining whether a redundant equivalent subexpression exists, and said means for eliminating said redundant equivalent subexpression.

14. (Canceled)

15. (Currently amended) The computer program product of claim 7, ~~whereby~~ wherein the simplifying rules in step (b) comprise:

(ba) arranging token pairs into subgroups;

(bb) arranging operand tokens in an arranged subgroup in order;

(bc) reducing the ordered operands by consolidating one or more constants and eliminating variables of opposite effect to form reduced subgroups; and

(bd) consolidating one or more multiple instances of similar subgroups, to produce a reduced string.

16. (Currently amended) The computer program product of claim 7, ~~whereby~~ wherein an algebraic expression whose equivalence is to be determined contains an aliased variable, said

computer program further comprising the steps of:

arranging steps for arranging an ordered list of aliases of the variable, and substituting steps for substituting a first alias in the ordered list for all instances of the aliased variable in the expression, wherein said compiling process steps for compiling comprise said arranging steps for arranging an ordered list of aliases.

17. (Currently amended) The computer program product of claim 7, ~~whereby~~ wherein an algebraic expression whose equivalence is to be determined contains a function, said computer program further comprising the steps of:

reducing steps for reducing function arguments using the set of predetermined simplifying rules; and

replacing steps for replacing the function by a tagged string, said string designating a function name, parameter types, and arguments, ~~whereby~~ wherein the tag distinguishes the function name from a variable, wherein said compiling process steps for compiling comprise said reducing steps for reducing function arguments and said replacing steps for replacing the function by a tagged string.

18. (Previously presented) The computer program product of claim 7, wherein said reduction process steps comprise:

processing steps for processing token pairs into an ordered arrangement;

determining steps for determining whether a redundant equivalent subexpression exists within said ordered arrangement; and

eliminating steps for eliminating said redundant equivalent subexpression from said ordered arrangement.

19. (Canceled)